Message based Models and Environments for Software Evolution

Rushikesh K. Joshi

Department of Computer Science & Engineering

Indian Institute of Technology Bombay

rkj@cse.iitb.ac.in

Some recent paradigms for (product) aspect capture

- Classes, objects, is-a relation, association and part-whole relations, components, component composition
- Concurrency: Threads and synchronizers
- Distribution: Interfaces loosely coupled from implementation
- Interoperability: IDLs and tools

Concerns and their programming expressions

- Can separate concerns be expressed separately and be traceable eventually?

- If the answer is yes→
  - Independent development processes for the concerns
  - Independent abstractions (representations)
  - Proceed to integration mechanisms

- Limitations of popular OOP
  - Many cross-cutting concerns posed problem
    - Design reuse vs. implementation reuse
    - Transparencies for system/context variabilities
    - Generic concerns covering multiple abstractions
  - Rise of new paradigms: AOP extensions
    - Context relations, compositional filters, aspectJ

A Static (Popular) Solution

- Aspect specifications separate from base specifications
- Aspects weaved with bases by a Weaver
- Static (compile time) weaving
- Weaved code looses aspects → no traceability of aspects into first class runtime elements of the language

Limitations of aspects

- Implementation-centric approach
- Lacks Process
- Non-first class core – static approach
- Contracts may get violated

Some other static approaches

- Overloading, Template classes to more powerful generic specification languages
  - Generic (e.g. XML based) specification applied to base code which is transformed

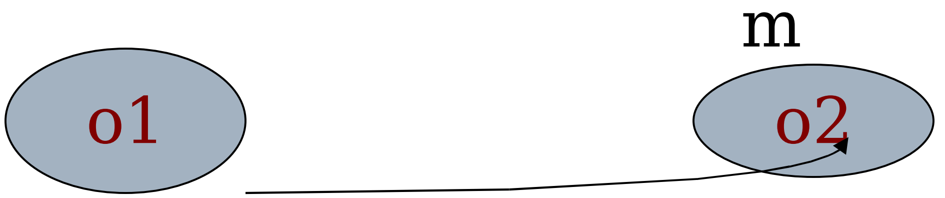
Some Dynamic Approaches

- Subclassing and Polymorphism
- Using Metalevel protocols: e.g. Smalltalk's metaclasses
- Reflection into PL implementations

Our approach: Capturing Dynamics through Communications Abstractions (First class filter objects)

- Honor encapsulation, target messaging
- Aspects are first class entities in base language: objects with member functions and local state
- Abstractions for Specification are same as those of base language
- Weaving is replaced by pairing at runtime
  - At object level

An Interaction



Entities(Objects/Components)
Message generation
Message flow
Message delivery
Methods
Response flow
Response delivery

Targeting messaging, leaving objects as they are

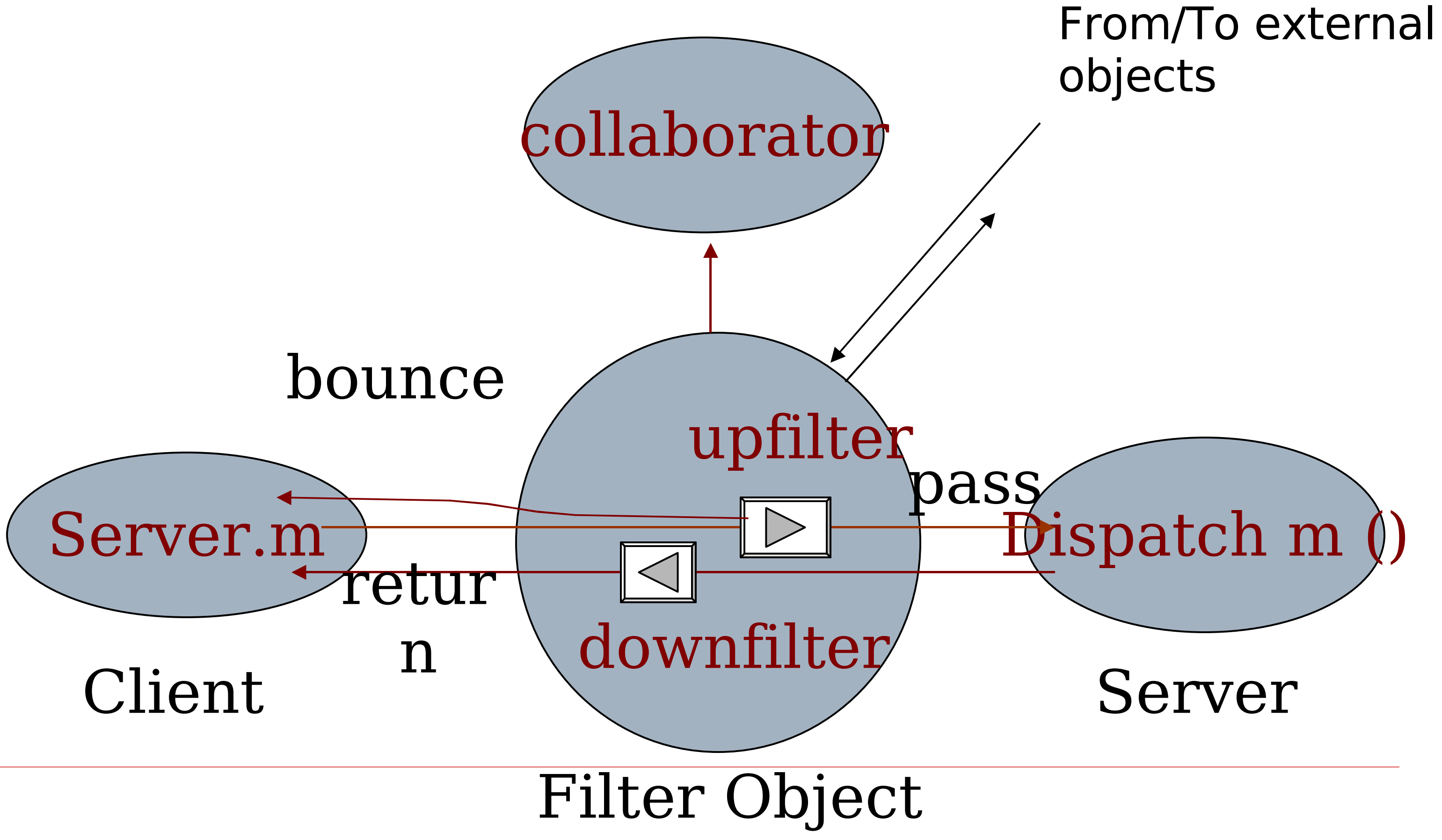
Entities (Objects/Components)
Message generation
Message flow
Message delivery
Methods
Response flow
Response delivery

Separation of Message Processing from Message Control

- *Message Processing*
  - Determines response by the receiver once the message is dispatched to the receiver
  - Implementation of the component/object’s contract
- *Message Control*
  - Activities on/over messages in transit
    - i.e. during information flow

An Example Paradigm

Primitives:
Process
Forward
Replace
Force
Delay
Bounce



Class level specification

```
Class Dictionary {
    ...
}

Class Cache: filter Dictionary {


    ....
}
```

Instance level pairing (not weaving)

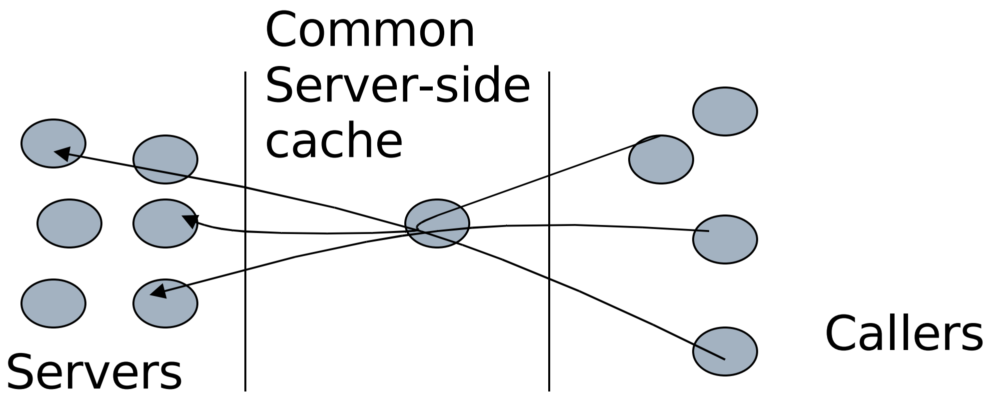
plug and *unplug*

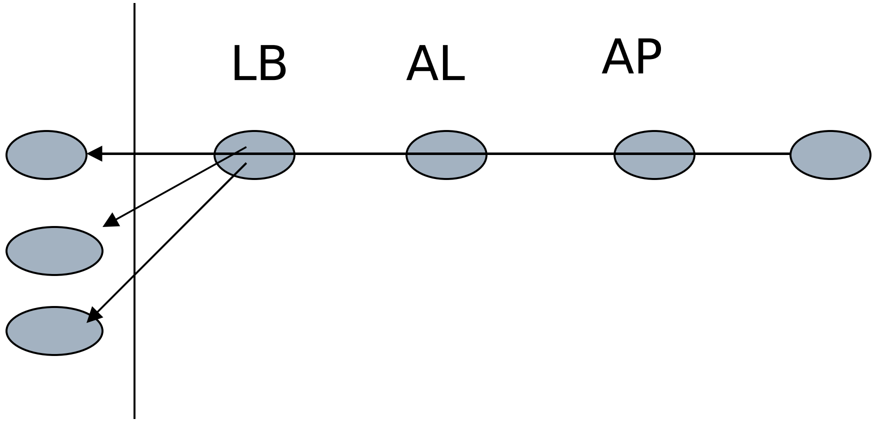
```
main ( ) {
Dictionary *d=..;
Cache *c=..;
    plug d c;
    ...
    unplug d;
}
```

First class representation in an OOPL

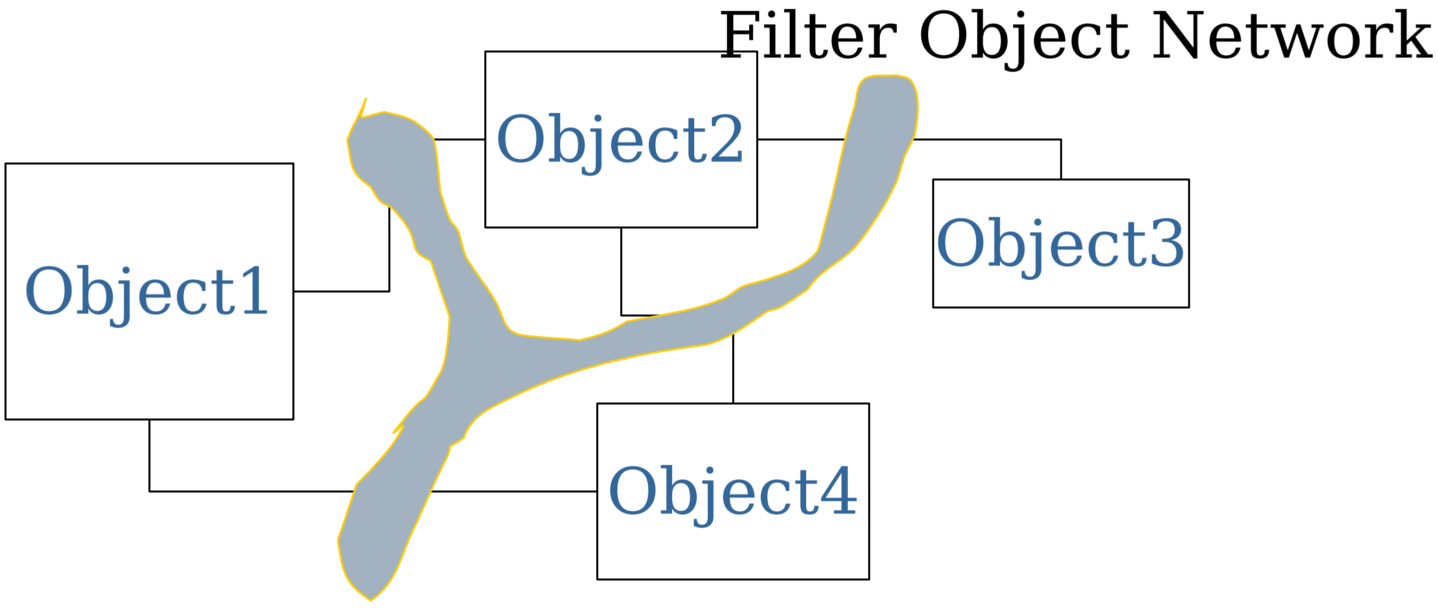
```
Class Dictionary {
    public: Meaning SearchWord( Word);
}
class Cache : filter Dictionary {
upfilter:
    Meaning SearchCache(Word) filters SearchWord;
downfilter:
    Meaning ReplaceCacheEntry (Meaning) filters SearchWord;
public:
    double hitRatio ( );
private:
        … implementation
}
```

Dynamic Grouping and Layering





Orthogonal Collaborative Frameworks: Crosscutting functionalities



Patterns at Messaging Layer

- Message replacement
- Receiver Replacement
- Routing, destination selection
- Repeater
- Message Content Replacement (value transformer)
- Decoration (logger)
- Message hold/delay and synchronize

Replacer

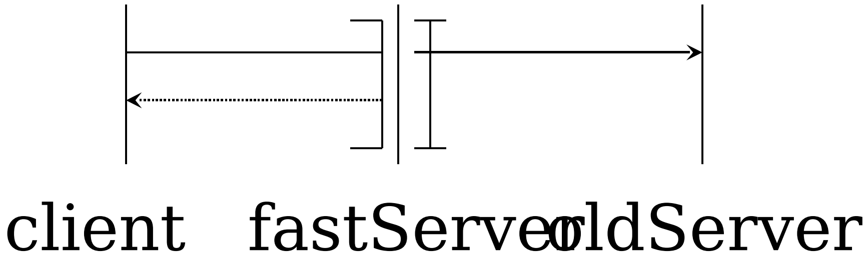
- A filter member function operates as a replacement function to its corresponding server member function

FastServer | oldServer =

filter interface:

    funcReplacer (in) upfilters oldServer :: func (in)

        = [v <-- self.func (in); bounce (v);]



Router

- A filter member function operates as a router function

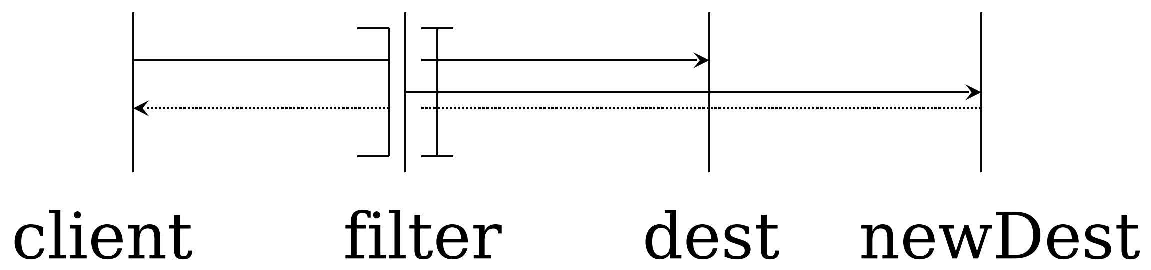
balancer | searchEngine =

filter interface:

    searchRouter (item) upfilters SearchEngine ::search (item)

        = [newDest <-- self.nextDest();

            v<--newDest.search(item); bounce (v);]



Repeater

- A filter member function dispatches the filtered invocation to multiple servers

enrollFilter | centralEnroller =

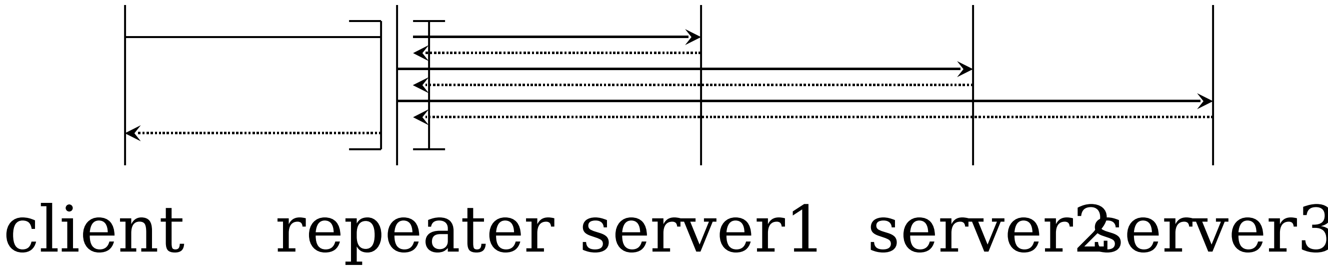
filter interface:

libEnroll (student) upfilters centralEnroller :: enroll (student)

= [if (student.dept == civil) civilLib-->enroll (student);

if (student.status == minor)minorBody-->
enroll(student);

pass;]



Approaches for Middleware

- Need-to-filter principle: A server is declared as *Filterable Server*

```
interface Filterable {
attach (in Object filter)
detach ();
};
interface Server : Filterable {
service ();
}
```

- *Filter Object aware Middleware (e.g. MICO extensions)*

Dynamic Functional Evolution
(functional cross-cut)

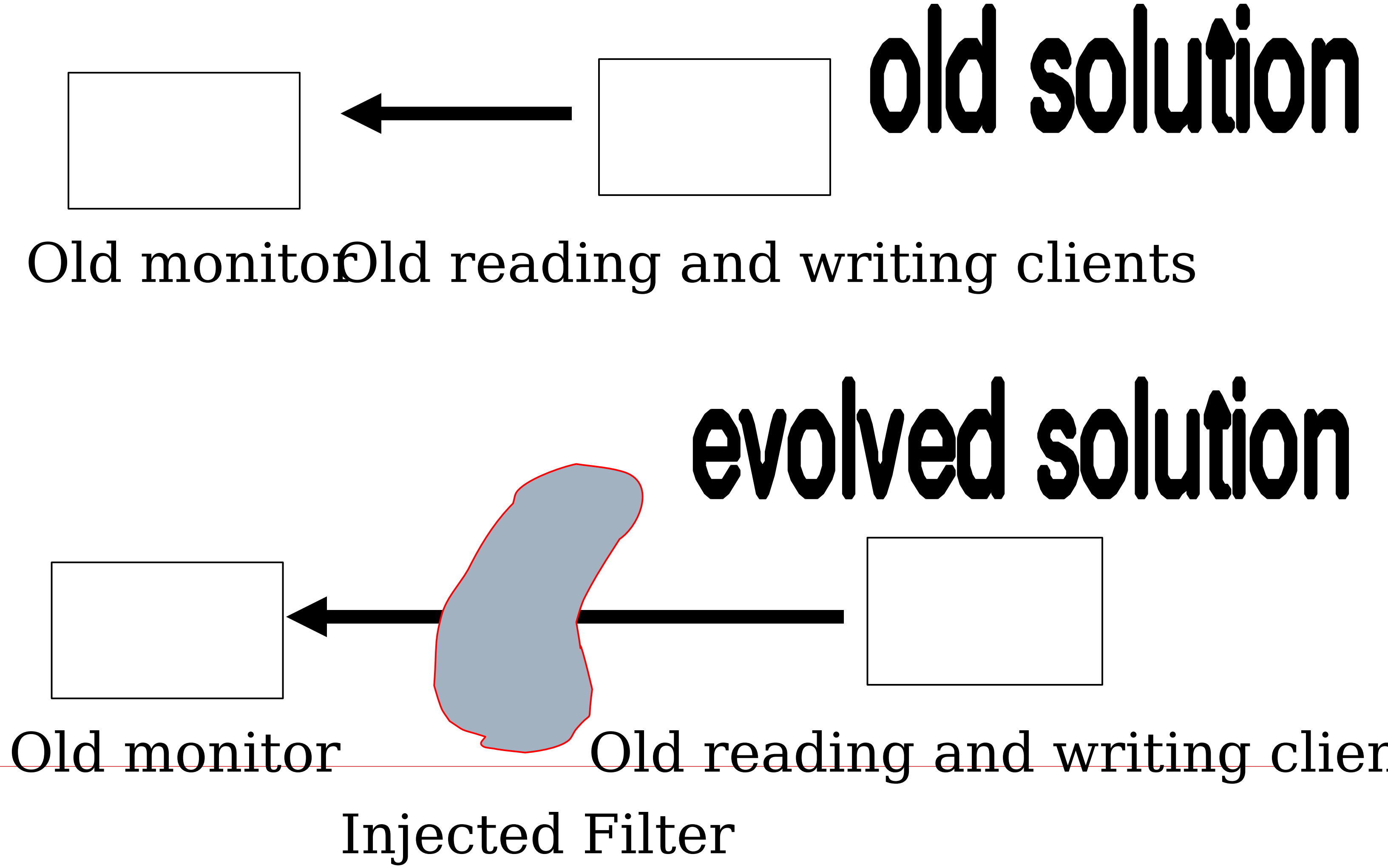
- A Readers and Writers Solution
  - (Hansen 1978)

```
process resource
s: int
proc StartRead   when s>0 : s++; end
proc EndRead     if s >1: s--; end
proc StartWrite  when s==1: s--; end
proc EndWrite    if s==0: s++; end
s=1;
```

Evolution Requirement

- Solve the same problem with additional constraint that further reader requests should be delayed as long as there are writers waiting or using the resource

The Approach



Evolution using Filter Processes

```
process problemSolver: filter resource
www : int
upfilter:
      SW_Ufilter filters StartWrite
      SR_Ufilter filters StartRead
downfilter:
      EW_Dfilter filters EndWrite
proc SW_Ufilter: www++; pass; end
proc EW_Dfilter: www---; end
proc SR_Ufilter:          when www==0: pass; end
www=0;
```

State of the Art

1. Models for C++/COM components

3. TJF (Translator for Java Filters)

5. Middleware: MICO kernel extensions
   Filter aware middleware

7. Implementations of Filter Objects in Distributed Systems over AspectJ+RMI

9. Distributed Filter Processes (Unimplemented)

11. An interpreter of sigmaF calculus (an untyped abstract language)

Ongoing projects in this area in specific and evolution in general

- Semantics (Abadi/Cardelli style) and implementations
- Applications, Notations, Methods and Patterns
- Component Adapters
- Refactoring Techniques
- Support for Dynamic Evolution in Environments, and methodologies